A PROBLEM SIMPLIFICATION APPROACH THAT GENERATES HEURISTICS FOR CONSTRAINT-SATISFACTION PROBLEMS

Rina Dechter
Judea Pearl

April 1985
CSD-850014

A PROBLEM SIMPLIFICATION APPROACH THAT GENERATES HEURISTICS FOR CONSTRAINT-SATISFACTION PROBLEMS

by

Rina Dechter and Judea Pearl

Cognitive System Laboratory, Computer Science Department, U.C.L.A

ABSTRACT

Many AI tasks can be formulated as Constraint-Satisfaction problems (CSP), i.e., the assignment of values to variables subject to a set of constraints . Recognition of 3-dimensional objects, puzzles solving, electronic circuit analysis and truth-maintenance systems are examples of such problems, and these are normally solved by various versions of backtrack search. In this work we show how advice can be automatically generated to guide the order by which the search algorithm assigns values to the variables, so as to reduce the amount of backtracking. The advice is generated by consulting relaxed models of the subproblems created by each value-assignment candidate. The relaxed problems are chosen to yield a backtrack-free solutions, and the information retrieved from these models induces a preference order among the choices pending in the original problem.

We identify a class of CSP whose syntactic and semantic properties make them easy to solve. The syntactic properties involve the structure of the constraint graph while the semantic properties guarantee some local consistencies among the constraints. In particular, tree-like constraint graphs can be easily solved and are chosen therefore as the target model for the relaxation scheme. Optimal algorithms for solving easy problems are presented and analyzed. A scheme for constructing a "best" constraint-tree approximation to a given constraint graph is introduced and, finally, the utility of using the advise is evaluated in a synthetic domain of CSP problem instances.

1. BACKGROUND AND MOTIVATION

1.1 Introduction

An important component of human problem-solving expertise is the ability to use knowledge about solving easy problems to guide the solution of difficult ones. Only a few works in AI [Sacerdoti 1974, Carbonell 1983] have attempted to equip machines with similar capabilities. Gaschnig [Gaschnig 1979] Guida et al. [Guida 1979], and Pearl [Pearl 1983] suggested that knowledge about easy problems could be instrumental in the mechanical discovery of heuristics. Accordingly, it should be possible to manipulate the representation of a difficult problem until it is approximated by an easy one, solve the easy problem, then use the solution to guide the search process in the original problem.

The implementation of this scheme requires three major steps: 1. simplification 2. solution 3. advice generation. Additionally, to perform the simplification step, we must have a simple, **a-priori** criterion for deciding when a problem lends itself to easy solution.

This paper uses the domain of constraint-satisfaction tasks to examine the feasibility of these three steps. It establishes criteria for recognizing classes of easy problems, provides special procedures for solving them, demonstrate a scheme for generating good relaxed models, and introduces an efficient method for extracting advice from them. Finally, the utility of using the advice is evaluated in a synthetic domain of problem instances.

Constraint-satisfaction problems (CSP) involve the assignment of values to variables subject to a set of constraints. Understanding three-dimensional drawings, graph coloring, electronic circuit analysis, and truth maintenance systems are examples of CSPs. These are normally solved by some version of backtrack search which may require exponential search time (for example, the graph coloring problem is known to be NP-complete.)

The following paragraphs summarize the basic terminology of the theory of CSP as presented in [Montanari 1974] and extended by [Mackworth 1977] and [Freuder 1982]. Some observations are presented regarding the relationships between the representation of the problem and the performance of the backtrack algorithm.

1.2 Definitions and Nomenclature

Formally, the underlying model of a CSP involves a set of n variables $X_1, \ldots, X_n$ each having a set of domain values $D_1, \ldots, D_n$. An **n-ary relation** on these variables is a subset of the Cartesian product:

$$\rho \subseteq D_1 \times D_2 \times, \ldots, \times D_n . \tag{1}$$

A **binary constraint** R_{ij} between two variables is a subset of the Cartesian product of their domain values, i.e.,

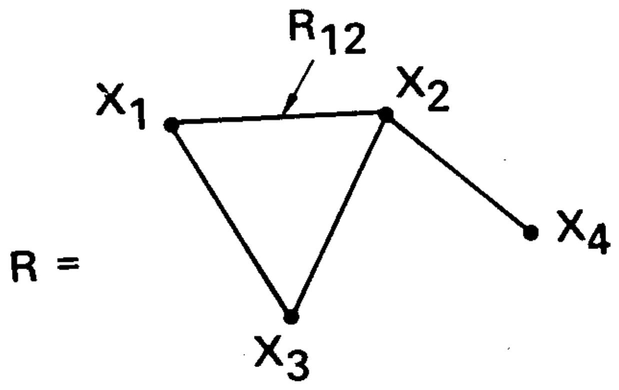
$$R_{ij} \subseteq D_i \times D_j . \tag{2}$$

A **network of binary constraints** is the set of variables $X_1, \ldots, X_n$ plus the set of binary constraints between pairs of variables and it represents an n-ary relation defined by the set of n-tuples that satisfy all the constraints. Formally, Given a symmetric network of constraints between n variables, the relation ρ represented by it is:

$$\rho = \{(x_1, x_2, \ldots, x_n) \mid x_i \in D_i, and\ (x_i, x_j) \in R_{ij} \text{ for all } ij\} . \tag{3}$$

Not every n-ary relation can be represented by a network of binary constraints with n variables, and the issues of finding the best approximation by such network are addressed in [Montanari 1974]. In this paper we will discuss only relations induced by network of binary constraints and henceforth assume that all constraints are binary and symmetric.

Each network of constraints can be represented by a **constraint graph** where the variables are represented by nodes and the non-universal constraints by arcs. The constraints themselves can be represented by the set of pairs they allow, or by a matrix in which rows and columns correspond to values of the two variables and the entries are 0 or 1 depending on whether the corresponding pair of values is allowed by the constraint. Figure 1 displays a typical network of constraints (a), where constraints are given using matrix notation (b).



(a)

$$\begin{array}{c|ccc} & a & b & c \\ \hline a & 0 & 1 & 0 \\ b & 1 & 0 & 1 \\ c & 0 & 1 & 0 \end{array}$$

(b)

FIGURE 1

Several operations on constraints can be defined. The useful ones are: union, intersection, and composition. The **union** of two constraints between two variables is a constraint that allows all pairs that are allowed by either one of them. The **intersection** of two constraints allows only pairs that are allowed by both constraints. The **composition** of two constraints, $R_{12}\ R_{23}$ "induces" a constraint R_{13} defined as follows: A pair (x_1,x_3) is allowed by R_{13} if there is at least one value $x_2 \in D_2$ such that $(x_1,x_2) \in R_{12}$ and $(x_2,x_3) \in R_{23}$. If matrix notation is used to represent constraints, then the induced constraint R_{13} can be obtained by matrix multiplication:

$$R_{13} = R_{12} \cdot R_{23} \tag{4}$$

A partial order among the constraints can be defined as follows: $R_{ij} \subseteq R'_{ij}$ iff every pair allowed by R_{ij} is also allowed by R'_{ij} (this is exactly set inclusion). In this case we say that R_{ij} is **smaller** (or stronger) than R'_{ij}. We can also say that R'_{ij} is a relaxation of R_{ij}. The smallest constraint between variables X_i and X_j is the **empty constraint**, denoted Φ_{ij}, which does not allow any pair of values. The largest (i.e. weakest) is the **universal constraint**, denoted U_{ij}, which permits all possible pairs. A corresponding partial order can be defined among networks of constraints having the same set of variables. We say that $R \subseteq R'$ if the partial order is satisfied for every pair of corresponding constraints in the networks.

Finally, we define the notion of equivalence among networks of constraints: two networks of constraints with the same set of variables are **equivalent** if they represent the same n-ary relation.

Consider, for example, the network of figure 2, representing a problem of four bi-valued variables. The constraints are attached to the arcs and are given, in this case, by sets of pairs. The direction of the arcs only indicates the way by which constraints are specified. The constraint between X_1 and X_4, displayed in part (b), can be induced by R_{12} and R_{24}. Therefore, adding this constraint to the network will result in an equivalent network. Similarly, since the constraint R_{21} can be induced from R_{23} and R_{31} it can be deleted without changing the relation represented by the network.

The process of inducing relations in a given network makes the constraints smaller and smaller, while leaving the networks equivalent to each other. Montanari called the smallest network of constraints which is equivalent to a given network R, **The Minimal Network**. The minimal network of constraints makes the "global" constraints on the network as "local" as possible. In other words, a minimal network of constraints is perfectly explicit.

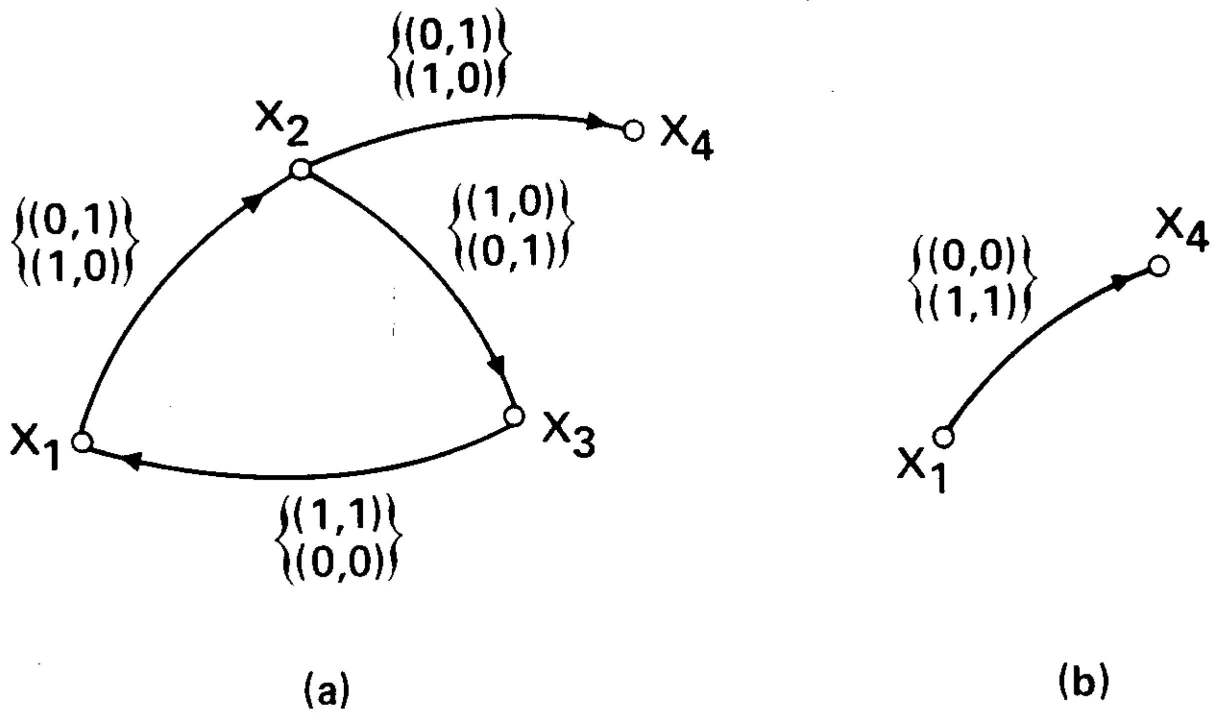


FIGURE 2

Every binary-constrained CSP problem can be represented by a network of constraints. A tuple in the relation represented by the network is called a solution. The problem is either to find all solutions, one solution, or to verify that a certain tuple is a solution. The last problem is fairly easy while the first two problems can be difficult and have attracted a substantial amount of research.

1.3 Backtrack for CSP

The algorithm mostly used to solve CSP problems is backtrack. Given a **vertical order** of the set of variables $X_1, X_2, \ldots, X_n$ and a **horizontal order** of values in each variable's domain $x_{i,1}, x_{i,2}, \ldots, x_{i,k}$, algorithm Backtrack for finding one solution is given below:

```
Backtrack
Begin
 1. Assign $x_{1,1}$ to $X_1$ (if allowed by a unary constraint)
 2. k=1
 3. while  k$\leq$ n-1
 4.    while $X_{k+1}$ has more values /*values $x_1,x_2, \ldots ,x_k$ were already selected*/
 5.      choose the first value $x_{k+1,j}$ of $X_{k+1}$, s.t. constraints($x_1,x_2, \ldots ,x_k,x_{k+1,j}$) = true
 6.      then   erase (temporarily) $x_{k+1,1}, \ldots ,x_{k+1,j}$ from domain of $X_{k+1}$
 7.             k=k+1
 8.             goto 3
 9.    end
 10.   k=k-1 (backtrack since no value at (5) exists).
 11.   If k=0 exit , no solution exists.
 12. end
 13. exit with solution
End.
```

In line 5 of the algorithm all the constraints between X_{k+1} and previous variables in the vertical order are checked. The value chosen should be consistent with all the previous instantiated values under those constraints. For Backtrack to find all solutions the above algorithm should be modified slightly by adding another outer loop and terminating only when k=0.

Montanari considered the question of finding the minimal network M of a given network R as the central problem in CSP, implying that once it is available the problem is virtually solved. The following two lemmas elaborate on this issue by relating the minimal network to the backtrack algorithm.

Lemma 1:

Let R and R' be two equivalent networks such that $R' \subseteq R$, then given the same order for instantiating variables, any sequence of values that is explicated by Backtrack on R' will be explicated also by Backtrack on R when Backtrack looks for all solutions.

Proof:

The order between the networks implies that any sequence of values which is consistent under R' is also consistent under R.

□

Conclusion:

Given a network R and a fixed order of variables' instantiation, Backtrack's performance, when looking for all solutions, is most efficient on the minimal network, relative to all networks which are equivalent to R, since it is contained in all of them.

We now show that when the algorithm seeks only one solution then, using the minimal network, the solution can be found easily in many cases. Some more definitions are required.

Given an n-ary relation ρ, representable by a network with n variables, the projection ρ_S of the relation ρ on a subset S of the variables is not always representable by a network with |S| nodes. If for any subset of variables, S, ρ_S is representable by a network with |S| variables then ρ is said to be a **Decomposable relation**. Given an n-ary decomposable relation ρ, represented by a minimal network M, then for any subset S of variables the subnetwork of M restricted to the nodes in S, is a minimal network of ρ_S. In this case M is also said to be decomposable.

For example, the network in figure 3 is minimal but not decomposable. The relation represented by M is:

$$\rho = \{(x_{1,1},x_{2,1},x_{3,1},x_{4,1}),(x_{1,1},x_{2,2},x_{3,2},x_{4,2}),(x_{1,2},x_{2,2},x_{3,1},x_{4,3})\} \tag{5}$$

(Note that X_4 is a non-binary variable.)

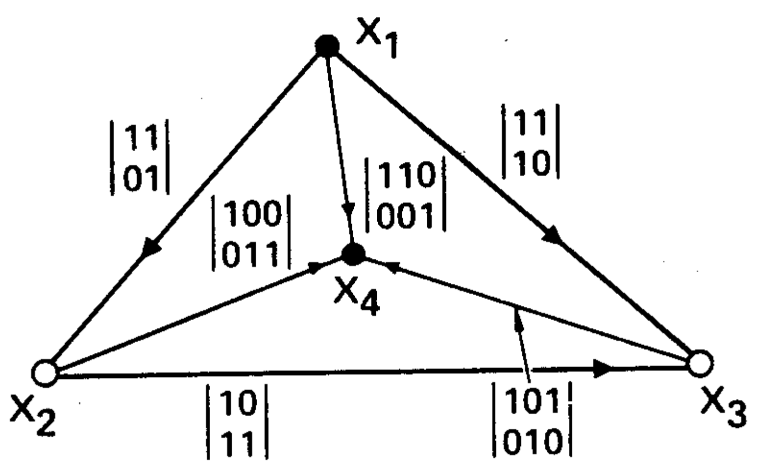
If S = $\{X_1,X_2,X_3\}$ it can be shown that ρ_s, given by

$$\rho_S = \{(x_{1,1},x_{2,1},x_{3,1}),(x_{1,1},x_{2,2},x_{3,2}),(x_{1,2},x_{2,2},x_{3,1})\}, \tag{6}$$

cannot be represented by a network with 3 variables. (For more details see [Montanari 1974]).

Lemma 2:

If M is minimal and decomposable network then Backtrack will find one solution without backtracking at all.



X_1
$\begin{vmatrix}11\\01\end{vmatrix}$
$\begin{vmatrix}11\\10\end{vmatrix}$
$\begin{vmatrix}110\\001\end{vmatrix}$
$\begin{vmatrix}100\\011\end{vmatrix}$
X_4
X_2
$\begin{vmatrix}10\\11\end{vmatrix}$
$\begin{vmatrix}101\\010\end{vmatrix}$
X_3

FIGURE 3

Proof:

From M's decomposability it follows that any projection ρ_s has a minimal network which is the subnetwork of M that is restricted to the variables in S. Therefore, any tuple of the variables in S that satisfy all the constraints in the minimal subnetwork is part of an n-tuple in the n-ary relation represented by M, and therefore it can always be extended.

□

The complexity of finding a solution given a minimal and decomposable network M is, therefore, $O(n^2 \cdot k)$ where n is the number of variables and k is the maximum cardinality of the value domain for all variables. In the previous example of a nondecomposable minimal network Backtrack may explore the path $x_{1,1}$, $x_{2,2}$, $x_{3,1}$ and since it cannot be extended to a 4-tuple relation satisfying M the algorithm will have to backtrack. In conclusion we see that solving a CSP problem , finding all or one solution, is easier when the minimal network is available, but this does not guarantee backtrack-free search unless the network is also decomposable.

Backtrack and its performance on CSP problems were extensively discussed in the AI literature. Most researchers have been trying to identify the major maladies in its performance, to provide a corresponding cure, and to analyze the results. These works can be classified along the following dimensions:

1. The problem objectives: finding all or finding one solution

2. control parameters: controlling the order of variables' instantiation, order of values' instantiation, or manipulating the problem's representation by pruning values or propagating constraints.

3. cure implementation: preprocessing the cures prior to the start of the algorithm, or incorporating them dynamically into the algorithm while it searches for solution(s).

Mentioning only few studies, we start with [Montanari 1974] who considered the task of finding all solutions, and discussed the solution of a problem by propagating the constraints and pruning pairs of values from them. In light of the previous lemmas these methods can be regarded as a preprocessing phase to a backtrack algorithm although the latter was not mentioned explicitly. Mackworth [Mackworth 1977] extended Montanati's work by introducing consistency checks to cure the maladies of Backtrack. Haralick and Eliot [Haralick 1980] discussed the task of finding all solutions and examined various methods of value pruning including lookahead mechanisms which are incorporated into the algorithm. Freuder [Freuder 1982] considered the problem of finding one solution to a CSP problem and provided a procedure to select a good ordering of variables which is performed as a preprocessing to Backtrack. Other works in analyzing the average performance of Backtrack were reported by [Nudel 1983, Purdom 1985] and [Haralick 1980] all estimating the size of the tree exposed by Backtrack while searching for all solutions.

It seems that the only parameter not considered for controlling Backtracks' performance is the order by which values are assigned to variables. Part of the reason can be explained by the following theorem.

Theorem 1:

Given the objective of finding all solutions and given a fixed vertical order for variables' instantiation, the search tree exposed by Backtrack is invariant to the order of values' selection. (All search trees which are identical up to an ordering of branches are considered the same.)

Proof:

Any sequence of values that is explored by Backtrack w.r.t. a specific order of variables is consistent under this subset of variables, and it may or may not lead to a solution. The only way Backtrack can find out if it is extensible to a solution is to continue and explore it. Therefore, Backtrack which tries to find all solutions will have to search this sequence for any order of value assignment.

□

Similarly, Backtrack that looks for one solution, in a CSP that has no solution, will expose the same search tree under any order of value assignment, given a fix vertical order.

The above theorem states that value-selection strategies cannot be used to improve Backtrack's performance for the task of finding all solutions. In this paper we address the objective of finding a single solution to CSPs. Although this problem is easier it can still be very difficult (e.g. 3-colorability) and it appears frequently. Theorem proving, planning and even vision problems are examples of domains where finding one solution will normally suffice [Simon 1975], and, the order by which values are selected may have a profound effect on the algorithm's performance. In the following section we outline a general approach to devising value selection strategies.

1.4 General Approach for Automatic Advice Generation

Following the model of the A^* algorithm that uses heuristics to guide the selection of the next node for expansion, we now wish to guide Backtrack in selecting the next node on its path. We assume that the order of variables is fixed and therefore the selection of the next node amounts to choosing a promising assignment of values from a set of pending options. Clearly, if the next value can be guessed correctly, and if a solution exists, the problem will be solved in linear time with no backtracking. Backtrack builds partial solutions and extends them as long as they show promise to be part of a whole solution. When a dead-end is recognized it backtracks to a previous variable. The advice we wish to generate should order the candidates according to the confidence we have that they can be extended further to a solution.

Such confidence can be obtained by making simplifying assumptions about the continuing portion of the search graph and estimating the likelihood that it will contain a solution even when the simplifying assumptions are removed. It is reasonable to assume that if the simplifying assumptions are not too severe then the number of solutions found in the simplified version of the problem would correlate positively with the number of solutions present in the original version. We, therefore, propose to count the number of solutions in the simplified model and use it as a measure of confidence that the options considered will lead to an overall solution.

To incorporate the advice generation into the backtrack algorithm, line 5 should be replaced by the following:

```
5a. eliminate all values of $X_{k+1}$ which are not consistent with $x_1, \ldots, x_k$.
5b. /* let $x_{k+1,1}, \ldots, x_{k+1,t}$ all the remaining candidates for assignment*/
  advise(($x_{k+1,1}, \ldots, x_{k+1,t}$), ($x'_{k+1,1}, \ldots, x'_{k+1,t}$))
5c. assign $x'_{k+1,1}$ to $X_{k+1}$
```

The **advise** procedure takes the set of consistent values of X_{k+1} and order them according to the estimates of the number of possible solutions stemming from them.

The remaining sections describe the advice-giving algorithm, provide theoretical grounds for it, and report experimental evaluation of its performance. In section 2 we establish criteria for recognizing classes of easy CSP problems and introduce an efficient method of counting the number of solutions. Section 3 describes a process of approximating a given CSP problem by an easy relaxed one. Section 4 evaluates the utility of using the advice using a synthetic domain of CSP problems.

2. THE ANATOMY OF EASY CONSTRAINT-SATISFACTION PROBLEMS

2.1 Introduction and background

In general, a problem is considered easy when its representation permits a solution in polynomial time. However, since we are dealing mainly with backtrack algorithms, we will consider a CSP **easy** if it can be solved by a **backtrack-free** procedure. A backtrack-free search is one in which Backtrack terminates without backtracking, thus producing a solution in time linear with the number of variables.

The discussion of backtrack-free CSPs relies heavily on the concept of constraint graphs. Freuder [Freuder 1982] has identified sufficient conditions for a constraint graph to yield a backtrack-free CSP, and has shown, for example, that tree-like constraint graphs can be made to satisfy these conditions, with a small amount of preprocessing. Our main purpose here is to further study classes of constraint graphs lending themselves to backtrack-free solutions and to devise efficient algorithms for solving them. Once these classes are identified they can be chosen as targets for a problem simplification scheme: constraints can be selectively deleted from the original specification so as to transform the original problem into a backtrack-free one. As already mentioned, we propose to use the "number of consistent solutions in the simplified problem" as a figure-of-merit to establish priority of value assignments in the backtracking search of the original problem. We show that this figure of merit can be computed in time comparable to that of finding a single solution to an easy problem.

Definition: ([Freuder 1982]) **An ordered constraint graph** is a constraint graph in which the nodes are linearly ordered to reflect the sequence of variable assignments executed by the

Backtrack search algorithm. The **width of a node** is the number of arcs that lead from that node to previous nodes, the **width of an ordering** is the maximum width of all nodes, and the **width of a graph** is the minimum width of all the orderings of that graph.

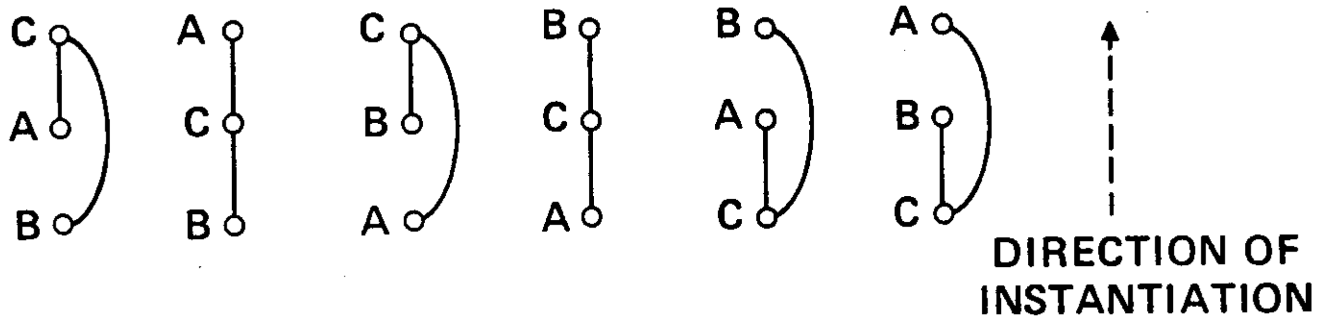


FIGURE 4

Figure 4 presents six possible orderings of a constraint graph. The width of node C in the first ordering (from the left) is 2, while in the second ordering it is 1. The width of the first ordering is 2, while that of the second is 1. The width of the constraint graph is, therefore, 1. Freuder provided an efficient algorithm for finding both the width of a graph and the ordering corresponding to this width. He further showed that a constraint graph is a tree iff it is of width 1.

Montanari [Montanari 1974] and Mackworth [Mackworth 1977] have introduced two kinds of local consistencies among constraints named **arc consistency** and **path consistency**. Their definitions assume that the graph is directed, i.e., each symmetric constraint is represented by two directed arcs.

Let $R_{ij}(x,y)$ stand for the assertion that (x,y) is permitted by the explicit constraint R_{ij}.
Definition: ([Mackworth 1977]): Directed arc (X_i, X_j) is **arc consistent** iff for any value x $\in$

D_i there is a value y $\in D_j$ such that $R_{ij}(x,y)$.

Definition ([Montanari 1974]): A path of length m through nodes $(i_0,i_1, \ldots ,i_m)$ is **path consistent** if for any value $x \in D_{i_0}$ and $y \in D_{i_m}$ such that $R_{i_0 i_m}(x,y)$, there is a sequence of values $z_1 \in D_{i_1}, \ldots , z_{m-1} \in D_{i_{m-1}}$ such that

$$R_{i_0 i_1}(x,z_1) \text{ and } R_{i_1 i_2}(z_1,z_2) \text{ and } \cdots \; R_{i_{m-1} i_m}(z_{m-1},y). \tag{7}$$

$R_{i_0 i_m}$ may also be the universal relation e.g., permitting all possible pairs.

A constraint graph is arc (path) consistent if each of its directed arcs (paths) is arc (path) consistent. Achieving "arc consistency" means deleting certain values from the domains of certain variables such that the resultant graph will be arc-consistent, while still representing the same overall set of solutions. To achieve path-consistency, certain pairs of values that were initially allowed by the local constraints should be disallowed. Montanari and Mackworth have proposed polynomial-time algorithms for achieving arc-consistency and path consistency. In [Mackworth 1984] it is shown that arc consistency can be achieved in o(ek^3) while path consistency can be achieved in o(n^3k^5), where n is the number of variables, k is the number of possible values, and e is the number of edges.

Theorem 2([Freuder 1982]):

a. If the constraint graph has a width 1 (i.e. the constraint graph is a tree) and if it is arc consistent then it admits backtrack-free solutions.

b. If the width of the constraint graph is 2 and it is also path consistent then it admits backtrack-free solutions.

□

The above theorem suggests that tree-like CSPs (CSPs whose constraint graph are trees) can be solved by first achieving arc consistency and then instantiating the variables in an order which makes the graph exhibit width 1. Since this backtrack-free istantiation takes

$O(e \cdot k)$ steps, and on trees e=n-1, the whole problem can be solved in $O(nk^3)$. The test for this property is also easily verified: to check whether a given graph is a tree can be done by a regular $O(n^2)$ spanning tree algorithm. Thus, tree-like CSPs are easy.

The second part of the theorem tempts us to conclude that a width-2 constraint graph should admit a backtrack-free solution after passing through a path-consistency algorithm. In this case, however, the path-consistency algorithm may add arcs to the graph and increase its width beyond 2. This often happens when the algorithm deletes value-pairs from a pair of variables that were initially related by the universal constraint (having no connecting arc between them), and it is often the case that passage through a path-consistency algorithm renders the constraint graph complete. It may happen, therefore, that no advantage could be taken of the fact that a CSP possesses a width-2 constraint graph if it is not already path consistent. We are not even sure whether width-2 suffices to preclude NP-completeness.

In the following section we give weaker definitions of arc and path consistency which are also sufficient for guaranteeing backtrack-free solutions but have two advantages over those defined by [Montanari 1974] and [Mackworth 1977] :

1. They can be achieved more efficiently, and
2. They add fewer arcs to the constraint-graph, thus preserving the graph width in a larger classes of problems.

2.2 Algorithms for achieving directional consistency

The Case of Width-1

Securing full arc-consistency is more than is actually required for enabling backtrack-free solutions in constraint graphs which are trees. For example, if the constraint graph in figure 5 is ordered by (X_1,X_2,X_3,X_4), nothing is gained by making the directed arc (X_3,X_1) consistent.

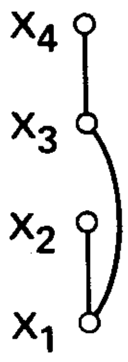


FIGURE 5

To ensure backtrack-free assignment, we need only make sure that any value assigned to variable X_1 will have at least one consistent value in D_3. This can be achieved by making only the directed arc (X_1,X_3) consistent, regardless of whether (X_3,X_1) is consistent or not. We, therefore, see that arc-consistency is required only w.r.t a single direction, the one specified by the order in which Backtrack will later choose variables for instantiations. This motivates the following definitions.

Definition: Given an order d on the constraint graph R, we say that R is **d-arc-consistent** if all the directed edges which follow the order d are arc-consistent.

Theorem 3:

Let d be a width-1 order of a constraint tree T. If T is d-arc-consistent then the backtrack search along the order d is backtrack-free.

proof:

Suppose that $X_1,X_2, \ldots ,X_k$ were already instantiated. The variable X_{k+1} is connected to at most one previous variable (from the width-1 property), say X_i, which was assigned the value x_i. Since the directed arc (X_i,X_{k+1}) is along the order d, its arc-consistency implies the existence of a value x_{k+1} such that the pair (x_i,x_{k+1}) is permitted by the constraint $R_{i(k+1)}$. Thus, the assignment of x_{k+1} is consistent with all previous assignments.

□

An algorithm for achieving directional arc-consistency for any ordered constraint graph is given next (The order d $=(X_1,X_2,\ldots,X_n)$ is assumed)

DAC- d-arc-consistency

```
1. begin
2.   For i=n to 1 by -1  do
3.     For each arc $(X_j,X_i)$; j < i do
4.        REVISE($X_j,X_i$)
5.     end
6.   end
7. end
```

The algorithm REVISE(X_j,X_i), given in [Mackworth 1977] , deletes values from the domain D_j until the directed arc (X_j,X_i) is arc-consistent.

REVISE(X_j,X_i)

```
1. begin
2.   For each x∈$D_j$ do
3.     if there is no value y∈$D_i$ such that $R_{ji}(x,y)$ then
4.          delete x from $D_j$
5.   end
6. end
```

To prove that the algorithm achieves d-arc-consistency we have to show that upon termination, any arc (X_j,X_i) along d ($j < i$), is arc-consistent. The algorithm revises each d-directed arc once. It remains to be shown that the consistency of an already processed arc is not violated by the processing of coming arcs. Let arc (X_j,X_i) ($j < i$) be an arc just processed by REVISE(X_j,X_i). to destroy the consistency of (X_j,X_i) some values should be deleted from the domain of X_i during the continuation of the algorithm. However, according to the order by which REVISE is performed from this point on, only lower indexed variables may have their set of values updated. Therefore, once a directed arc is made arc-consistent its consistency will not be violated.

The algorithm AC-3 [Mackworth 1977] that achieves full arc-consistency is given for reference:

```
AC-3
1. begin
2.   Q <- { ($X_i$,$X_j$) | ($X_i$,$X_j$) ∈ arcs, i≠ j}
3.     while Q is not empty do
           select and delete arc ($X_k$,$X_m$) from Q
5.         REVISE($X_k$,$X_m$)
6.         if REVISE($X_k$,$X_m$) caused any change then
7.            Q <- Q ∪ {($X_i$,$X_k$) | ($X_i$,$X_k$) ∈ arcs ,i≠k,m}
7.     end
8. end
```

The complexity of AC-3, is O(ek^3), while the directional arc-consistency algorithm takes ek^2 steps since the REVISE algorithm, taking k^2 tests, is applied to every arc exactly once. It is also optimal, because even to verify directional arc-consistency each arc should be inspected once, and that takes k^2 tests. Note that when the constraint graph is a tree, the complexity of the directional arc-consistency algorithm is O(nk^2).

Theorem 4:

A tree-like CSP can be solved in O(nk^2) steps and this is optimal.

proof:

Given that we know that the constraint graph is a tree, finding an order that will render it of width-1 takes O(n) steps. A width-1 tree-CSP can be made d-arc-consistent in O($n \cdot k^2$) steps, using the DAC algorithm. Finally, the backtrack-free solution on the resultant tree is found in O($n \cdot k$) steps. Summing up, finding a solution to tree-like CSP's takes, O($n \cdot k$) + O(nk^2) + O(n) = O(nk^2). This complexity is also optimal since any algorithm for solving a tree-like problem must examine each constraint at least once, and each such examination may take in the worst case k^2 (especially when no solution exist and the constraints permit very few pairs of values).

□

Interestingly, if we apply DAC w.r.t. order d and then DAC w.r.t. the reverse order we get a full arc-consistency for trees. We can, therefore, achieve full arc-consistency on trees in O(nk^2). Algorithm AC-3, on the other hand, can be shown to have a worst case performance on trees of O(nk^3). On general graphs, however, we shell next show that (full) arc-consistency cannot be achieved in less then $e \cdot k^3$ steps and, therefore, the the AC-3 algorithm is optimal.

Theorem 5:

A lower bound for achieving (full) arc-consistency on graphs is $\Omega(e \cdot k^3)$.

Proof:

We present a problem instance that cannot be made arc-consistent in less then $e \cdot k^3$. The problem, has n variables connected in a cycle, as shown in figure 6. The connections between the variables will be described for the 3-element network but can be easily extended to any number of variables. Variable X has k values, Variables Y, and Z, have k+1 values each. The constraint from X to Y maps values in X to values in Y which are incremented by 1. The constraints between Y and Z and between Z and X are both the equality mapping, except that k+1 of Z is mapped to k of X. The inconsistent arc is (Y,X) since the value 0 of Y has no pair in X. Removing 0 from D_Y makes the arc (Z,Y) inconsistent. This arc is examined and 0 is deleted, which make now the arc (X,Z) inconsistent and so on. Since we assume that any examination of an arc is an O(k^2) operation, and since at most one value is deleted from an arc while it is examined, each arc will be examined k-1 times, (there is just one solution: X=k, Y=k+1, Z=k+1), and the complexity in this case is $\Omega(n \cdot k^3)$. Therefore, in general achieving arc-consistency is $\Omega(e \cdot k^3)$.

□

Returning to our primary aim of studying easy problems, we now show how advice can be generated for solving a CSP using a tree-like approximation. Suppose that we want to solve an n variables CSP using Backtrack with $X_1, X_2, \ldots, X_n$ as the order of instantiation. Let X_i be the variable to instantiate next, with $x_{i1}, x_{i2}, \ldots, x_{ik}$ the possible candidate values. To

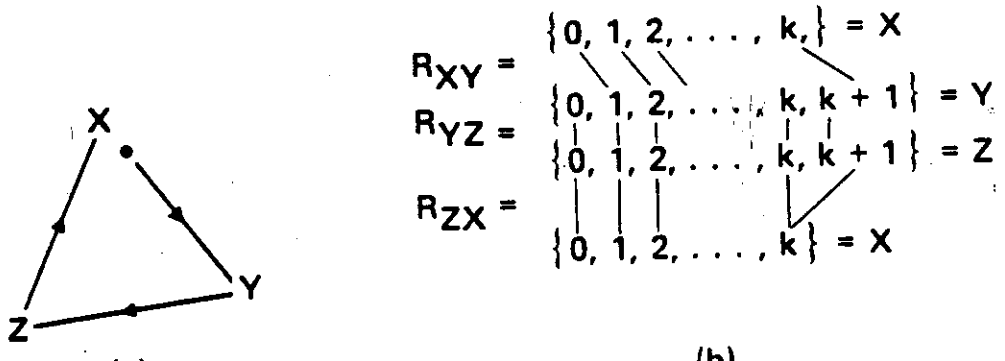

X
Y
Z
$R_{XY} =$
$R_{YZ} =$
$R_{ZX} =$
$\{0, 1, 2, \ldots, k,\} = X$
$\{0, 1, 2, \ldots, k, k+1\} = Y$
$\{0, 1, 2, \ldots, k, k+1\} = Z$
$\{0, 1, 2, \ldots, k\} = X$
(a)
(b)

FIGURE 6

minimize backtracking we should first try values which are likely to lead to a consistent solution but, since this likelihood is not known in advance, we may estimate it by counting the number of consistent solutions that each candidate admits in some relaxed problem. We generate a relaxed tree-like problem by deleting some of the explicit constraints given, then count the number of solutions consistent with each of the k possible assignments, and finally use these counts as a figure of merit for scheduling the various assignments. In the following we show how the counting of consistent solutions can be imbeded within the d-arc-consistency algorithm, DAC, on trees.

Any width-1 order, d, on a constraint tree determines a directed tree in which a parent always precedes its children in d (arcs are directed from the parent to its children). Let $N(x_{jt})$ stand for the number of solutions in the subtree rooted at X_j consistent with the assignment of x_{jt} to X_j. It can be shown that $N(\cdot)$ satisfy the following recurrence:

$$N(x_{jt}) = \prod_{\{c \mid X_c \text{ is a child of } X_j\}} \sum_{\{x_{cl} \in D_c \mid R_{jc}(x_{jt}, x_{cl})\}} N(x_{cl}) \qquad (8)$$

From this recurrence it is clear that the computation of N(x_{it}) may follow the exact same steps as in DAC; Simultaneously with testing that a given value x_{jt} is consistent with each of its children nodes, we simply transfer from each child of X_j to x_{jt} the sum total of the counts computed for the child's values that are consistent with x_{jt}. The overall value of N(x_{jt}) will be computed later on by multiplying together the summations obtained from each of the children. Thus, counting the number of solutions in a tree with n variables takes O(nk^2), the same as establishing directional arc-consistency.

The Case of Width-2

Order information can also facilitate backtrack-free search on width-2 problems by making path-consistency algorithms directional.

Montanari had shown that if a network of constraints is consistent w.r.t all paths of length 2 (in the complete network) then it is path-consistent. Similarly we will show that directional path-consistency w.r.t. length-2 paths is sufficient to obtain a backtrack-free search on a width-2 problems.

Definition: A constraint graph, R, ordered w.r.t. order d = (X_1,X_2, . . . ,X_n), is **d-path-consistent** if for every pair of values (x,y), x$\in X_i$ and y$\in X_j$ such that $R_{ij}(x,y)$ and i<j, there exists a value z$\in X_k$, k>j such that $R_{ik}(x,z)$ and $R_{kj}(z,y)$ for every k > i,j

Theorem 6:

Let d be a width-2 order of a constraint graph. If R is directional arc- and path-consistent w.r.t d then it is backtrack-free.

Proof:

To ensure that a width-2 ordered constraint graph will be backtrack-free it is required that the next variable to be instantiated will have values that are consistent with previously chosen values. Suppose that $X_1, X_2, \ldots, X_k$ were already instantiated. The width-2 property implies that variable X_{k+1} is connected to at most two previous variables If it is connected to X_i and X_j, $i,j \leq k$ then directional path consistency implies that for any assignment of values to X_i, X_j there exists a consistent assignment for X_{k+1}. If X_{k+1} is connected to one previous variable, then directional arc-consistency ensure the existence of a consistent assignment.

□

An algorithm for achieving directional path-consistency on any ordered graph will have to manage not only the changes made to the constraints but also the changes made to the graph, i.e., the arcs which are added to it. To describe the algorithm we use the matrix representation for constraints. The matrix R_{ii} whose off-diagonal values are 0, represents the set of values permitted for variable X_i. The algorithm is described using the operations of intersection and composition:
The intersection R_{ij} of R'_{ij} and R''_{ij} is written: $R_{ij} = R'_{ij} \,\&\, R''_{ij}$.

Given a network of constraints R = (V,E) and an order d = $(X_1, X_2, \ldots, X_n)$, we next describe an algorithm which achieves path-consistency w.r.t. this order.

```
DPC-d-path-consistency
  begin
(1) $Y^0 = R$
(2) for k= n to 1 by -1 do
      (a) $\forall$ i≤k connected to k do
            $Y_{ii} = Y^0_{ii} \,\&\, Y_{ik} \cdot Y_{kk} \cdot Y_{ki}$ /* this is REVISE(i,k)
      (b) $\forall$ i,j ≤ k s.t. $(X_i,X_k),(X_j,X_k) \in E$ do
            $Y_{ij} = Y_{ij} \,\&\, Y_{ik} \cdot Y_{kk} \cdot Y_{kj}$
            E <- E $\cup$ $(X_i,X_j)$
    end
  end
```

Step 2a is the equivalent of the REVISE(i,k) procedure, and it performs the directional arc-consistency. Step 2b updates the constraints between pairs of variables transmitted by a third variable which is higher in the order d. If X_i, X_j, i,j < k are not connected to X_k then the relation between the first two variables is not effected by X_k at all. If only one variable, X_i, is connected to X_k, the effect of X_k on the constraint (X_i,X_j) will be computed by step 2a of the algorithm. The only time a variable X_k effects the constraints between pairs of earlier variables is when it is connected to both. It is in this case only that a new arc may be added to the graph.

The complexity of the directional-path-consistency algorithm is $O(n^3k^3)$. For variable X_i the number of times the inner loop, 2b, is executed is at most $O((i-1)^2)$ (the number of different pairs less then i), and each step is of order k^3. The computation of loop 2a is completely dominated by the computation of 2b, and can be ignored. Therefore, the overall complexity is

$$\sum_{i=2}^{n}(i-1)^2k^3 = O(n^3k^3) \tag{9}$$

Applying directional-path-consistency to a width-2 graph may increase its width and therefore, does not guarantee backtrack-free solutions. Consequently it is useful to define the following subclass of width-2 CSP problems.

Definition: A constraint graph is **regular width-2** if there exist a width-2 ordering of the graph which remains width-2 after applying d-path-consistency, DPC.

A ring constitutes an example of a regular-width-2. Figure 7 shows an ordering of a ring's nodes and the graph resulting from applying the DPC algorithm to the ring. Both graphs are of width-2.

Theorem 7:

A regular width-2 CSP can be solved in $O(n^3k^3)$

Proof:

Regular width-2 problem can be solved by first applying the DPC algorithm and then performing a backtrack-free search on the resulting graph. The first takes $O(n^3k^3)$ steps and the second $O(e \cdot k)$ steps.

□

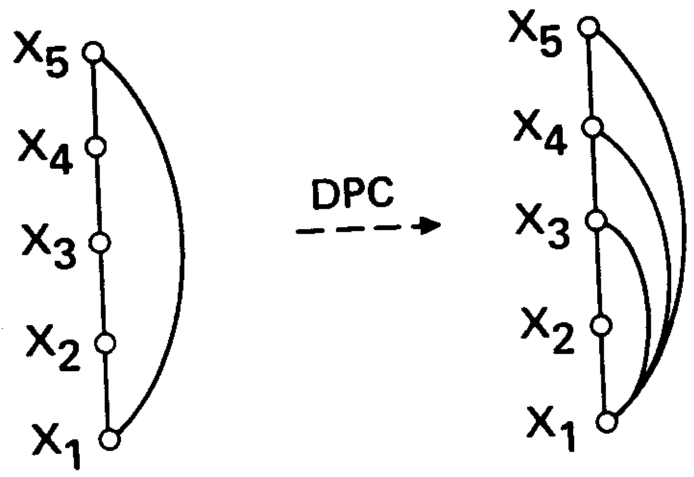


FIGURE 7

The main problem with the preceding approach is whether a regular width-2 CSP can be recognized from the properties of its constraint graph. One promising approach is to identify nonseparable components of the graph and all its separation vertices [Even 1979].

definition: A connected graph G(V,E) is said to have a **separation vertex** v if there exist vertices a and b, such that all the paths connecting a and b pass through v. A graph which has a separation vertex is called **separable**, and one which has none is called **nonseparable**. A subgraph with no separation vertices is called a **nonseparable component**.

An O(|E|) algorithm for finding all the nonseparable components and the separation vertices is given in [Even 1979]. It is also shown that the connectivity structure between the nonseparable components and the separation vertices has a tree structure.

Let R be a graph and SR be the tree in which the nonseparable components $C_1, C_2, \ldots, C_r$ and the separating vertices $V_1, V_2, \ldots, V_t$ are represented by nodes. A width-1 ordering of SR dictates a partial order on R, d^*, in which each separating vertex precedes all the vertices in its children components of SR.

Theorem 8:

If there exist a d^* ordering on R such that each nonseparable component is regular-width-2 then the total ordering is regular width-2.

Proof:

Let $d_C^\bullet$ be the order induced by $d^\bullet$ on component C, and let P_C be its parent separating vertex. When algorithm DPC reaches a node X, which is not P_C, within this component, then if X is not a separating vertex, it has arcs leading back only to nodes within this component. If the X is a separating vertex, then since it is not the parent of component C, all his children nodes were already processed and, therefore, it has arcs leading back only to nodes within its parent component C. In both cases DPC adds arcs only within the component C. Therefore, if $d^\bullet$ induces an order $d_C^\bullet$ which is regular-width-2 for all components, then $d^\bullet$ is regular-width-2.

□

As a corollary of theorem 8,we conclude that a tree of simple rings is regular-width-2. In figure 8, a graph with 10 nodes identified by its components and its separating vertices is given, with a possible $d^\bullet$ ordering which, in this case, is regular-width-2.

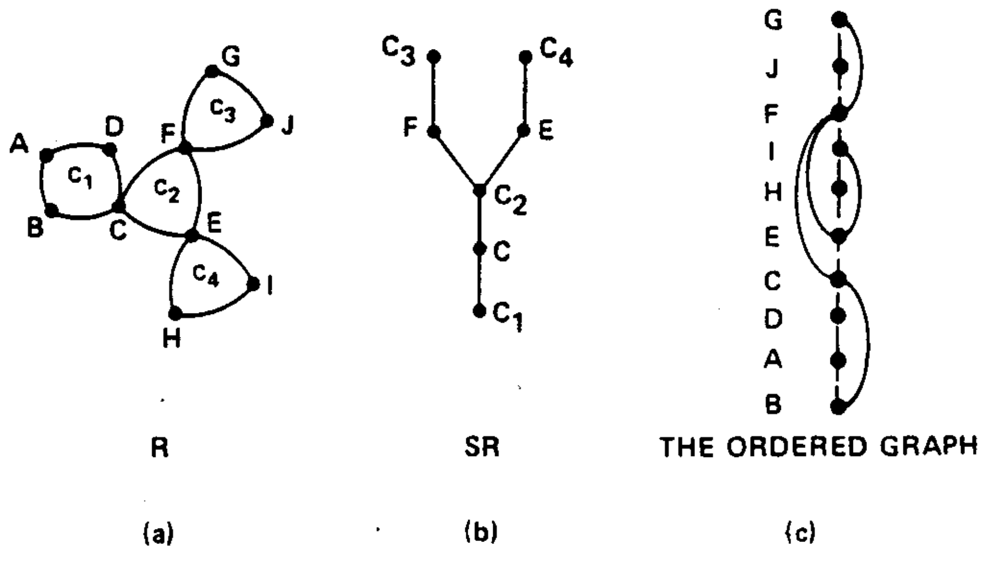


FIGURE 8

2.3 Summary and conclusions

Of the three main steps involved in the process of generating advise -- simplification, solution, and advice generation -- this section provides the following:

1. The simplification part: we have devised criteria for recognizing easy problems based on their underlying constraint graphs. The introduction of directionality into the notions of arc and path consistency enabled us to extend the class of recognizable easy problems beyond trees, to include regular width-2 problems.

2. The solution part: using directionality we were able to devise improved algorithms for solving simplified problems and to demonstrate their optimality. In particular, it is shown that tree-structured problems can be solved in $O(nk^2)$ steps, and regular width-2 problems in $O(n^3k^3)$ steps.

3. The advice generation part: we have demonstrated a simple method of extracting advice from easy problems to help Backtrack decide between pending options of value assignments. The method involves approximating the remaining part of a the task by a tree-structured problem, and counting the number of solutions consistent with each pending assignment. These counts can be obtained efficiently and can be used as figures-of-merit to rate the promise offered by each option.

3. THE SIMPLIFICATION PROCESS

The previous section suggests that a tree constraint-graph, being associated with an easy CSP, can be made a target to the simplification process from which advice will be extracted. We, therefore, discuss here the issues involved in approximating a network of binary constraints by a tree of constraints. We seek a good approximation since the closeness of the approximation tree to the original network will determine the reliability of the advice generated.

If the network R has an equivalent tree representation we would obviously like to recognize it and find such a representation. This, however may not be explicit in the constraint network; a network may contain many redundant constraints which, if eliminated, would still represent the same overall relation. For example, any one of the arcs in the network of figure 9 can be eliminated producing a tree-structured constraint graph representing the same relation. Note that in this figure, and throughout this section, there are multiple arcs between variables which connect values. Two values are connected if they are permitted by the constraint. Another example is given in figure 10 in which two 3-nodes networks, R_1 and R_2, are displayed. These two networks are equivalent, because they both represent the equality relation $\rho = \{(0,0,0),(1,1,1)\}$ and, unlike that of figure 9, both are maximal, that is, the addition of any pair of values to any one of the constraints (i.e relaxing any specific constraint) will result in a network representing a larger relation. Nevertheless, R_1 can be transformed into R_2 by simultaneously allowing the pair of values (1,0) between (Z,X) and disallowing the pair (0,1) between (X,Y). The question raised by this example is:

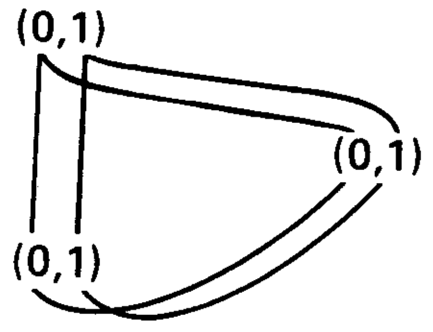


FIGURE 9

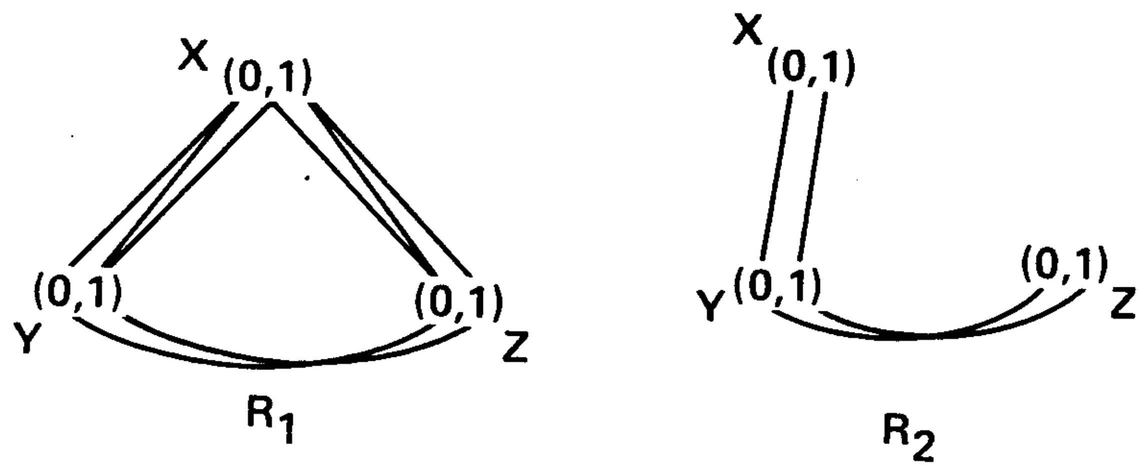


FIGURE 10

What networks have a tree representation and how to perform the transformation into a tree?

The two examples given display two levels of operation to be considered in the process of transforming a network into a tree. The first is a macro operation involving the deletion of whole arcs (i.e. total elimination of constraints between a pair of variables) while the second is a micro operation, that merely modifies the arcs by adding and deleting pairs of values. In our approach we will consider only macro operations of arc deletions; the use of micro transformations introduces a higher level of difficulty to which we will not relate at this point. Considering only arc deletions, a network R can be transformed into an equivalent tree only if some of the arcs are redundant, i.e. they represent constraints that can be inferred from others. This immediately raises the question of testing whether a given constraint is implied by others.

This question is the inverse of the oneposed by Montanari [Montanari 1974] who claimed that the central problem in Constraint-Satisfaction Problems is the transformation of the original network R into its minimal representation, M, which is the **most redundant** network that represents the same relation as R. Our interest is the opposite, transforming R into one of its least explicit equivalent network.

Definition:

a. A network R is **maximal** if there is no network R'; $R \subseteq R'$ such that R $\sim$ R'

b. A network R is **arc-maximal** if any arc deletion results in a network representing a larger relation.

A maximal network is arc-maximal but not necessarily vice-versa.

lemma 3:

An arc-consistent constraint tree is maximal.

proof:

In an arc-consistent tree, for any permitted pair of values there is an n-tuple in the relation which contains this pair. Disallowing this pair will eliminate such tuple from the relation, thus making the relation smaller. In other words any arc-consistent constraint tree is minimal network for that relation.

□

An immediate conclusion is that arc-consistent tree-network is arc-maximal. In general, deleting an arc from a tree-constraint may result in a larger relation even when it is not arc-consistent. Let T_1 and T_2 be the two disconnected subtrees generated from deleting arc (A,B) and let ρ_1 and ρ_2 be the projection of ρ on the variables in T_1 and T_2 respectively. The relation obtained after deleting the arc (A,B) from T is the product of ρ_1 and ρ_2 (i.e. any n-tuple that is the concatenation of a tuple in ρ_1 and a tuple in ρ_2). Therefore if there is a tuple in ρ_1 with A=a and a tuple in ρ_2 with B=b then the relation resulting from deleting arc (A,B) permits the pair (a,b).

In most cases a CSP problem will not be arc-redundant, because if it is posed by humans its specification has already passed through some process of redundancy filtering, and therefore arc deletion will almost always generate larger relations. The third question on which we will focus, therefore, is:
Given a network of constraints, R, what is the spanning tree, T, that will best approximate R?.

To discuss the quality of approximations, the notion of closeness between relations must first be agreed on. Let ρ be the relation represented by R and ρ_a the relation represented by a relaxed network R_a, i.e., $\rho \subseteq \rho_a$. An intuitively appealing measure for the closeness of R to R_a may be:

$$M(R,R_a) = \frac{|\rho|}{|\rho_a|} \tag{10}$$

where $|\rho|$ is the number of n-tuples in ρ. This measure satisfies:

a. $M(\rho,\rho) = 1$

b. If $\rho \subseteq \rho_a \subseteq \rho_b$ then

$$1 \geq M(R,R_a) \geq M(R,R_b)$$

M is a global property of two relations and the task of finding the spanning tree which yields the lowest M is very complex. Instead we propose a greedy approach: at each step delete the least "valuable" arc which leaves the network connected, namely, the arc deleted keeps the resulting network closest to the original one. To pursue this approach we need to define a measure of constraint strength, called **weight**, for each arc, that will estimate the contribution of that arc to the overall relation. Let R be a network of constraints and R' the network after the arc (X,Y) was eliminated, i.e. the constraint between X and Y becomes the universal constraint. Let l and l' be the size of the relations represented by R and R' respectively. $n'(x_i,y_j)$ is the number of tuples in the relation represented by R' having X=x_i Y=y_j, $R'(X,Y)$ is the constraint induced by R' on the pair (X,Y), and **r(X,Y)** is the local constraint given between X and Y in R. The following is satisfied

$$l = l' - \sum_{(x_i,y_j)\in R(X,Y)'-r(X,Y)} n'(x_i,y_j) \tag{11}$$

therefore

$$\frac{l}{l'} = 1 - \sum_{(x_i,y_j)\in R(X,Y)'-r(X,Y)} \frac{n'(x_i,y_j)}{l'} \tag{12}$$

Since we have no way of knowing the quantities $n'(x,y)$ and the structure of the induced constraint R'(X,Y), we will estimate them both by a constant,c and R' respectively. This gives:

$$\frac{l}{l'} = 1 - \frac{c}{l'}|R' - r(X,Y)| \tag{13}$$

The only quantity we can actually examine is r(X,Y), therefore to maximize $\frac{l}{l'}$ the above formula suggests choosing the constraint r(X,Y) with the most number of allowed value-pairs. Our first measure of constraint-weight is, therefore, defined by:

$$m_1(X,Y) = |r(X,Y)| \tag{14}$$

For instance, the weight of the universal constraint is m_1(X,Y) $= k^2$, and if r(X,Y) $= \Phi$ then the weight becomes m_1(X,Y) $= 0$.

In what follows we develop another measure of constraint strength by adopting notions from probability and information theory and by showing that the problem of finding s constraints tree approximation can be partially mapped into the problem of finding a tree-structured joint probability distribution [Chow 1968].

n-ary relations and probability distributions.

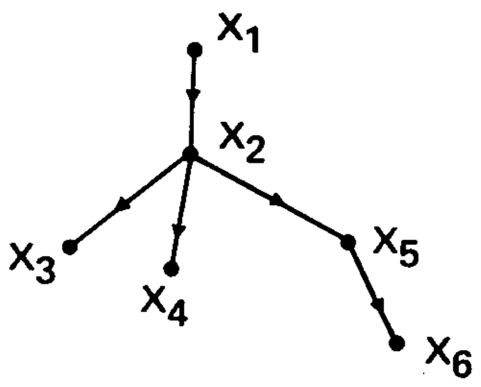
Let P(X) be a joint probability distribution of n discrete variables $X_1,X_2, \ldots ,X_n$. A product approximation of P(X) is defined to be a product of several lower-order distributions (also called marginal distributions) in such a way that the product is a probability extension of these lower order distributions. A particular class of product approximation considers only second order components where, each variable is conditioned upon at most one other variable. The relationships between the variables can be therefore represented by a tree. Given a directed spanning tree of the variables (the direction is from parents to sons) as in figure 11, the distribution function associated with it is given by the product:

$$P(X) = \prod_{X=(x_1,x_2, \ldots ,x_n)} P(x_i|x_{p(i)}) \tag{15}$$

$p(i)$ is the parent index of variable X_i, and P($x_0|x_{p(0)}$) = P(x_0) if 0 0 denotes the root of the tree. Chow [Chow 1968] has shown that if the measure of distance between two probability distributions P and P_a is given by:

$$I(P,P_a) = \sum_X P(X) \log \frac{P(X)}{P_a(X)} \tag{16}$$

then the closest tree-dependence distribution to P is the one that correspond to the maximum spanning tree when the weight of each arc is I(X_i,X_j). I(X_i,X_j) is Shanon's



$$P(X) = P(X_1) \cdot P(X_2 \mid X_1) \cdot P(X_3 \mid X_2) \cdot P(X_4 \mid X_2) \cdot P(X_5 \mid X_2) \cdot P(X_6 \mid X_5)$$

FIGURE 11

mutual information between X_i and X_j, defined by:

$$I(X_i,X_j) = \sum_{x_i,x_j} P(x_i,x_j)\log(\frac{P(x_i,x_j)}{P(x_i)P(x_j)} \tag{17}$$

$I(P,P_a)$ can be interpreted as the difference of the information contained in P(X) and that contained in $P_a(X)$ about P(X).

Chow's results are remarkable in that a global measure of closeness can be maximized by attending to local measures on individual arcs. We therefore, attempt to adopt chow's results to our need. Mapping probability distributions to constraints relations, we say that a relation ρ is associated with a distribution function P_ρ if:

$$P_\rho(x_1,x_2, \ldots ,x_n) = \begin{cases} 0 & \text{if } (x_1,x_2, \ldots ,x_n) \notin \rho \\ \dfrac{1}{|\rho|} & \textit{otherwise} \end{cases} \tag{18}$$

Let ρ_t be the relation represented by a constraint-tree,t, and let P_ρ and P_{ρ_t} be the distributions associated with relations ρ and relation ρ_t, having sizes of l and l_t, respectively. The "distance" between the two distributions:

$$I(P_\rho,P_{\rho_t}) = \sum_{X\in\rho} \frac{1}{l}\log\frac{l_t}{l} = \log\frac{l_t}{l} \tag{19}$$

is a monotone function of $\frac{l_t}{l}$ whose inverse was already proposed as a measure of closeness between two relations (where one contains the other). Accordingly, finding the closest tree dependence distribution P_{ρ_t} to P_ρ will result in the closest approximation of a tree relation ρ_t to ρ. Equivalently, in order to minimize $\frac{l_t}{l}$ we need to find the maximum spanning tree w.r.t the measure $I(X_i,X_j)$. From the given mapping between relations and distributions (Eq. (18)) we get that:

$$P(x_i,x_j)=\frac{n(x_i,x_j)}{l} \tag{20}$$

$$P(x_i)=\frac{n(x_i)}{l} \tag{21}$$

where $n(x_i,x_j)$ is the number of tuples in ρ having $X_i = x_i$ and $X_j = x_j$, and $n(x_i)$ is the number of tuples in ρ with $X_i = x_i$. Substituting (20) and (21) in (18) we get

$$I(X_i,X_j) = \sum_{x_i,x_j} \frac{n(x_i,x_j)}{l}\log l\cdot\frac{n(x_i,x_j)}{n(x_i)n(x_j)} \tag{22}$$

$$= \log l + \frac{1}{l}\sum_{x_i,x_j} n(x_i,x_j)\log\frac{n(x_i,x_j)}{n(x_i)n(x_j)} \tag{23}$$

Consequently the appropriate measure of arc weight is:

$$m(X_i,X_j) = \sum_{x_i,x_j} n(x_i,x_j)\log\frac{n(x_i,x_j)}{n(x_i)n(x_j)} \tag{24}$$

The question now is how to obtain the quantities $n(x_i)$, $n(x_i,x_j)$ needed for computing m. To find them accurately, we need to inspect the list of tuples permitted by the global relation which, of course, is unavailable. In the case of probability distributions the marginal probabilities $P(x_i)$, and $P(x_i,x_j)$ are estimated by sampling vectors from the distribution and calculating the appropriate sample frequencies. This cannot be done in our case since finding even one tuple that satisfies the network solve the entire problem. All that we have available is the network of constraints and, therefore, we must approximate the weight m(X,Y) by examining only properties of the arc (X,Y). This leads to approximations:

$$\hat{n}(x_i,x_j) = \begin{cases} 1 & (x_i,x_j) \in r(X_i,X_j) \\ 0 & otherwise \end{cases} \tag{25}$$

$$\hat{n}(x_i) = N_{X_j}(x_i) \tag{26}$$

Where $N_{X_j}(x_i)$ is the number of pairs in the constraint $r(X_i,X_j)$ with $X_i = x_i$, Substituting (25) and (26) in (24) we get:

$$m_2(X_i,X_j) = \sum_{(x_i,x_j)\in r(X_i,X_j)} \log \frac{1}{\hat{n}(x_i)\hat{n}(x_j)} \tag{27}$$

$$= - \sum_{(x_i,x_j)} (\log\hat{n}(x_i) + \log\hat{n}(x_j)) \tag{28}$$

$$= - \sum_{x_i}\hat{n}(x_i)\log\hat{n}(x_i) - \sum_{x_j}\hat{n}(x_j)\log\hat{n}(x_j) \tag{29}$$

The behavior of this measure can be illustrated in some special cases:

a. If the constraint r(X,Y) is the universal constraint (and assuming k values for each variable) then $m_2(u(X,Y)) = -2\sum(k-1)logk = -2k(k-1)logk$

b. If r(X,Y) is the empty constraint $\Phi(X,Y)$ then we define $m_2(\Phi(X,Y)) = 0$

c. If any value of X_i is allowed to go with exactly r values of X_j then $m_2 = -2k\cdot r\log r$. If r=1 we get $m_2 = 0$

d. when only one value in one variable is permitted with all the values of the other $m_2 = -k\cdot logk$

We see that this measure considers not only the number of the pairs allowed but also their distribution over the k^2 slots available. For uniform constraint (like case c) it can be seen that

$$m_2 = -2N \cdot \log r \tag{30}$$

when N is the size of the constraint.

We next give an example showing the behavior of the accurate measure of weight, m, compared with their estimates, m_2.

Consider the relation between 3 binary variables, X,Y,Z, given by:

$$\rho = \{(1,1,1),(1,0,0),(1,1,0),(0,0,0)\} \tag{31}$$

where the order of the variables is (X,Y,Z). A network representing this relation is given in figure 12 where the nodes are the variables and the lines correspond to permitted pair of values between pairs of variables.

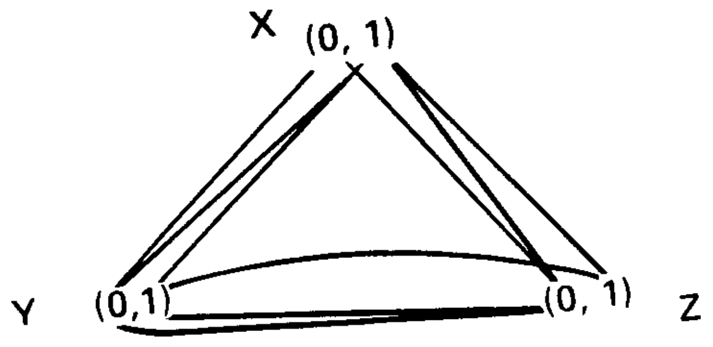


FIGURE 12

The accurate measures of $n(x_i,x_j)$, and, $n(x_i)$ for the pair (X,Y) are given by:

n(0,0)=1 , n(0,1)=0 , n(1,0)=1 , n(1,1)=2, n(X=0)=1, n(X=1)=3. Therefore, substituting in (26) we get:

$$m(X,Y)=\log\frac{1}{2} + \log\frac{1}{2\cdot 3}+2\log\frac{2}{3\cdot 2} =\log\frac{1}{108}$$

Similarly , for the two other pairs, we obtain:

$$m(X,Z) = \log \frac{4}{729}$$

$$m(Y,Z) = \log \frac{1}{108}$$

This suggest that the relation may be best approximated by a tree constituting of the arcs (X,Y) and (Y,Z). Indeed, the elimination of the arc (X,Z) will not change the relation at all whereas it is not possible to express ρ by removing either (Y,Z) or (X,Z) only.

By comparison, the network R and (32) give the weight estimates:

$m_2(X,Y) = -4$, $m_2(Y,Z) = -4$, $m_2(X,Z) = -4$

Which, in this case, fail to distinguish between the various constraints.

In conclusion, we suggest to generate tree-approximations for networks using the maximum spanning tree algorithm. Two measures for constraint-strength, to be used by the algorithm, are proposed and justified.

4. THE UTILITY OF THE ADVISE-GENERATION SCHEME

We compare here the performance of Advised Backtrack (abbreviated ABT) with that of Regular Backtrack (RBT) analytically, via worst case analysis, and experimentally, on a random constraint problem.

4.1 Worst case analysis

An upper bound is derived for the number of consistency checks performed by the algorithms as a function of the problem's parameters and the number of backtracks performed. A consistency check occurs each time the algorithm checks to verify whether or not a pair of values is consistent w.r.t. the corresponding constraint.

Let $\#B_A$ and $\#B_R$ be the number of backtracks, and N(ABT) and N(RBT) the number of consistency checks performed by ABT and RBT, respectively. The problem's parameters are n, the number of variables, and k, the number of values for each variable. Parameters associated with the constraint graph are |E|, the number of arcs, and deg, the maximum degree of variables in the graph.

The number of backtracks performed by an algorithm is equal to the number of leaves in the search tree which it explicates. We assume that

$$\text{Number of nodes expanded} = c\cdot\#B$$

approximately holds for some constant c. (This truly holds only for uniform trees where c is the branching factor.) Therefore we use the number of backtracks as a surrogate for the number of nodes expanded. Let $\#C_A$ and $\#C_R$ be the maximum number of consistency checks performed at each node by ABT and RBT, respectively. We have:

$$N \leq \#B \cdot \#C \tag{32}$$

Considering RBT first, the number of consistency checks performed at the i^{th} node in the order of instantiation is less then $k \cdot deg(i)$. That is, each of this variable's values should be checked against the previous assigned values for variables which are connected to it. We get:

$$N(RBT) \leq k \cdot deg \cdot \#B_R \tag{33}$$

The ABT algorithm performs all of its consistency checks within the advice generation. For the i^{th} variable , a tree of size $n-i$ is generated. The consistency checks performed on this tree occur in two phases. In the first phase, for each variable in the tree, the values which are consistent with the already assigned values are determined. The number of consistency checks for a variable v in the tree equal $k \cdot w(v)$, where $w(v)$ is the number of variables connected to v which were already instantiated. Therefore, for all variables in the tree, we have

$$k \cdot \sum_{v \in tree} w(v) \leq k \cdot |E| \ . \tag{34}$$

The second phase counts the number of solutions. We already showed that the counting takes no more then $(n-i) \cdot k^2$ which is bounded by nk^2. Hence,

$$N(ABT) \leq (k \cdot |E| + n \cdot k^2) \cdot \#B_A \tag{35}$$

We now want to determine the ratio between $\#B_A$ and $\#B_R$ for which it will be worthwhile to use Advised Backtrack instead of Regular Backtrack and, a first approximation, will treat the upper-bounds as tight estimates. If

$$N(ABT) \leq N(RBT) \tag{36}$$

then

$$(k \cdot |E| + n \cdot k^2) \cdot \#B_A \leq k \cdot deg \cdot \#B_R \ , \tag{37}$$

and therefore

$$\frac{\#B_R}{\#B_A} \geq \frac{|E|}{deg} + \frac{nk}{deg} . \tag{38}$$

Since

$$\frac{|E|}{deg} \leq n , \tag{39}$$

(38) will hold if

$$\frac{\#B_R}{\#B_A} \geq n + \frac{nk}{deg} . \tag{40}$$

Therefore, ABT is expected to result in a reduction in the number of consistency checks only if it reduces the number of backtracks by a factor greater than $(n+\frac{nk}{deg})$. Thus, the potential of the proposed method is greater in problems where the number of backtrackings is exponential in the problem size.

4.2 Experimental results

Test cases were generated using a random Constraint-Satisfaction Problem Generator. The CSP generator accepts four parameters: the number of variables n, the number of values for each variable k, the probability p_1 of having a constraint (an arc) between any pair of variables, and the probability p_2 that a constraint allows a given pair of values. Two performance measures were recorded: the number of backtrackings ($\#B$) and the number of consistency checks performed. The latter being an indicator of the overall running time of the algorithm. What we expect to see is that the more difficult the problems, the larger the benefits resulting from using advised Backtrack.

In our experiments we use m_1, the size of the constraint, as the weight for finding the minimal spanning tree. Using the alternative weight, m_2, is not expected to improve the results for two reasons. First, the problems generated were quite homogeneous and we have shown that for such problems both weights are the same. Second, the reduction in the number of backtrackings was so drastic that further improvements due to modified weights seems unlikely.

Two classes of problems were tested. The first, containing 10 variables and 5 values, were generated using $p_1 = p_2$=0.5, and the second with 15 variables and 5 values, generated using p_1=0.5 and p_2=0.6. 10 problems from each class were generated and solved by both ABT and RBT. The order by which the variables were instantiated was determined, for both algorithms, by the structure of the constraint graph. Namely, variables were selected in decreasing order of their degrees which closely correlate with the criterion of minimum width. [Freuder 1982] The order of value selection is determined by the advice mechanism in ABT and random in RBT. Therefore, while ABT solved each problem instance just once, RBT was used to solve each problem several (five) times to account for the variation in value selection order. When a problem has no solution, the number of backtrackings and consistency checks in RBT is independent on the order of value selection, and in these cases the problem was solved only once by RBT.

Figure 13 and figure 14 display performance comparisons for both classes of problems. In figure 13, the horizontal axis gives the number of backtrackings that were performed by RBT and the vertical axis gives the number of backtrackings performed by ABT for the same problem instance. The darkened circles correspond to problem instances from the first class while empty circles correspond to instances of the second class. We observe an impressive saving in $\#B$ when advise is used, especially for the second class in which the problems are larger. Figure 14 uses the same method to compare the number of consistency checks. Here, we observe that in many instances the number of consistency checks in ABT is larger than in RBT, indicating that in these cases the extra effort spent in "advising" backtrack, was not worthwhile.

These results are consistent with the theoretical prediction of the preceding subsection. If we substitute the parameters of the first class of problems in (42) we get that $\#B_A$ should be smaller than $\#B_R$ by at least a factor of 20 (25 for the second class of problems) to yield an improvement in overall performance. Many of the problems, however, were not hard enough (in terms of the number of backtrackings required by RBT) to achieve these levels.

Figure 15 compares the two algorithms in only those problems that turned out to be difficult. it displays the number of consistency checks in the cases where the number of backtrackings in RBT were at least 70. We see that the majority of these problems were solved more efficiently by ABT than RBT.

Experiments were also performed on the n-queen problem for n between 6 and 15 and on the 3-colorability problem on a set of random graphs. In all cases the number of backtrackings of ABT was smaller than RBT, but the problems were not difficult enough to get a net reduction in the number of consistency checks.

Experiments related to the ones reported here were performed by Haralick et al. [Haralick 1980]. The Forward-Checking lookahead mechanism (reported to exhibit the best performance considering the number of consistency checks) can also be viewed as an automatically generated advice in the sense discussed here. However, since the task was to find all solutions, the results cannot be directly related.

In conclusion, advice should be invoked on problems which are hard for RBT and, therefore, one needs a way of recognizing when a problem instance is difficult. For example, Knuth [Knuth 1975] has suggested a simple sampling technique that require very small computation to estimate the size of the search tree. These estimates can be used in conjunction with a parametrized advice generation that adapt itself according to the expected size of the tree. Namely, smaller problems will be guided by a weaker form of advice (e.g. based on partial trees) that is obtained more efficiently. Indeed in [Dechter 1985] we show experimentally that weaker advise is sufficient for reducing the amount of backtracking, therefore resulting in a more efficient search altogether.

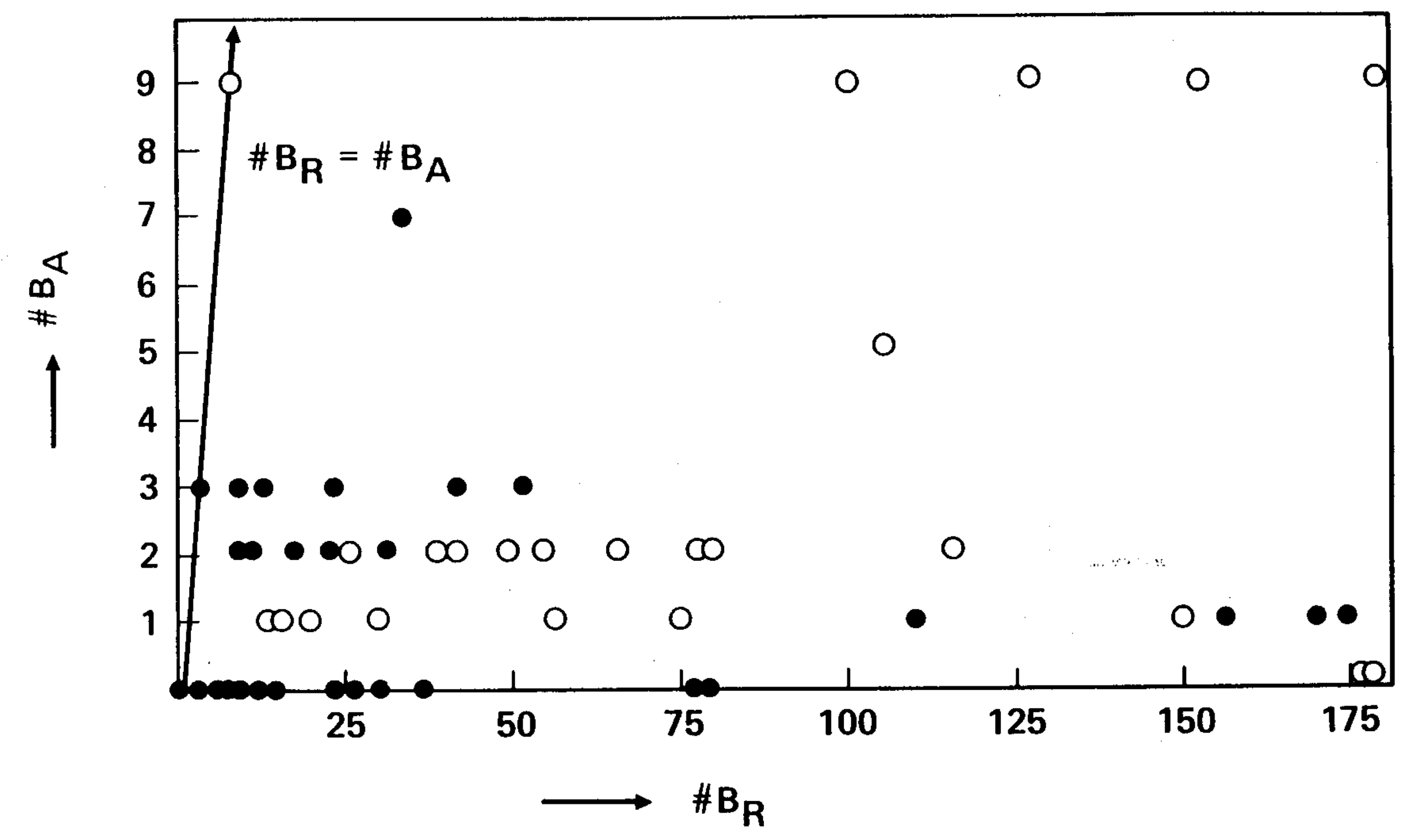
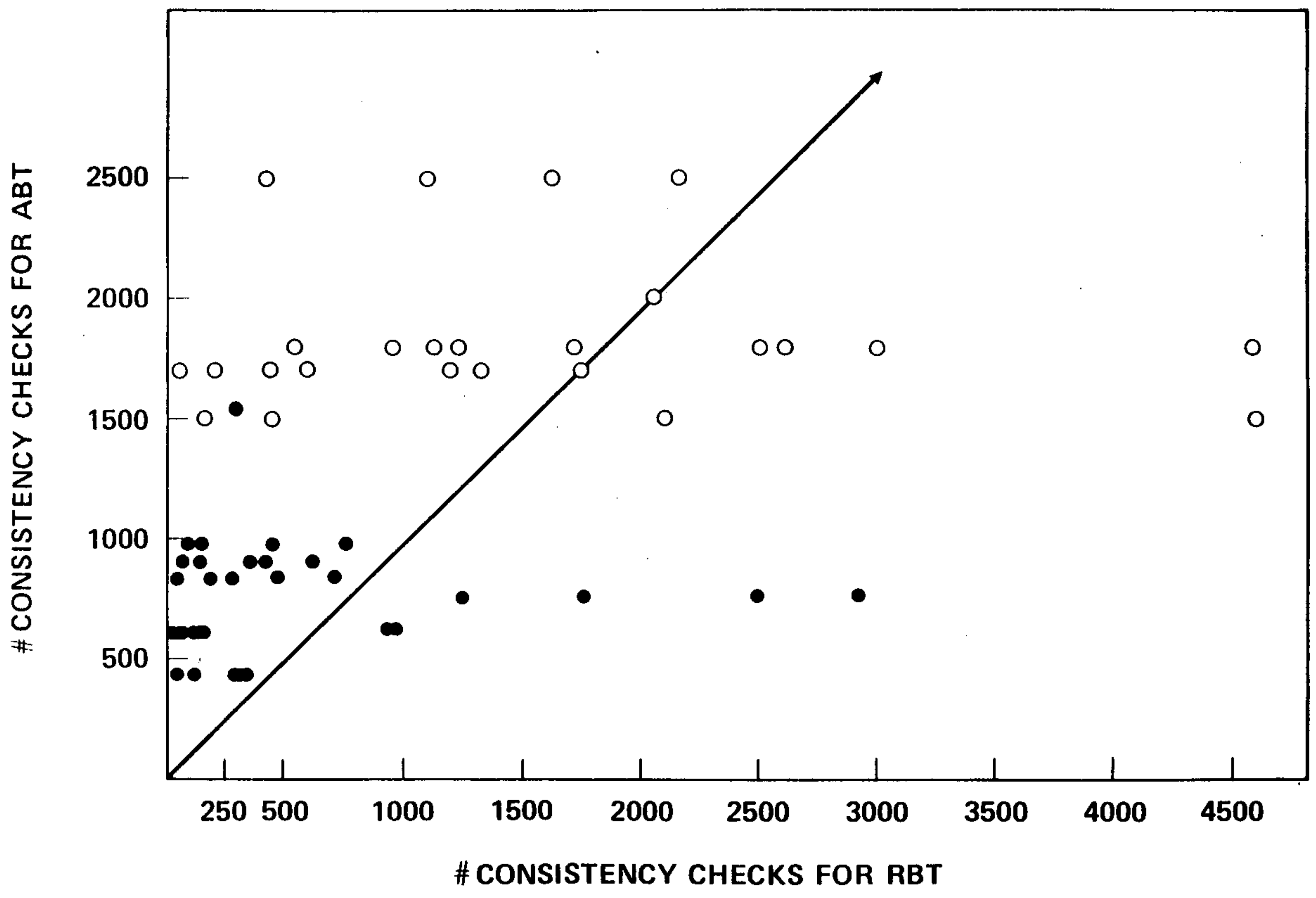


FIGURE 13



2500
2000
1500
1000
500
250
500
1000
1500
2000
2500
3000
3500
4000
4500
# CONSISTENCY CHECKS FOR ABT
# CONSISTENCY CHECKS FOR RBT

FIGURE 14

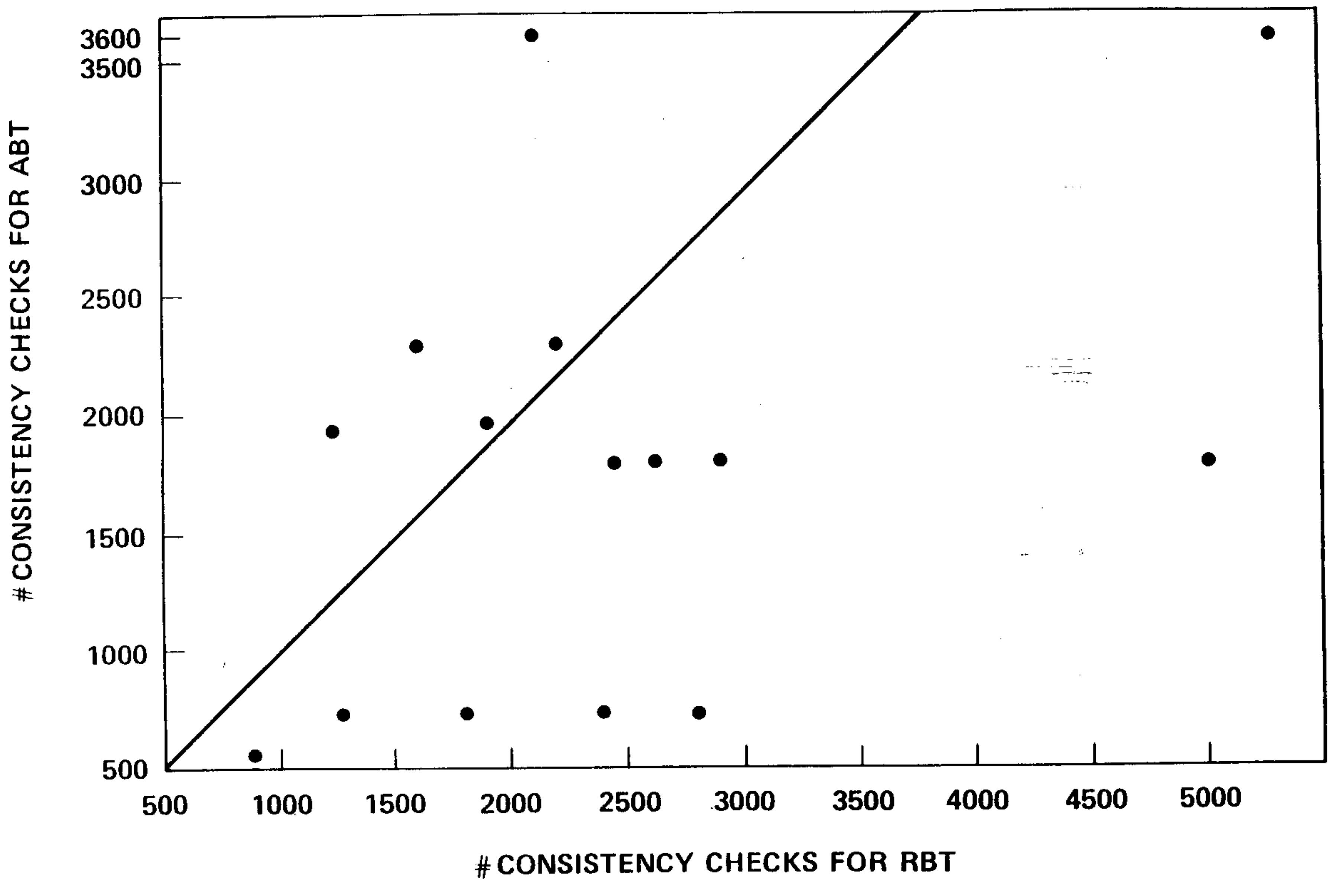

3600
3500
3000
2500
2000
1500
1000
500
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
# CONSISTENCY CHECKS FOR ABT
# CONSISTENCY CHECKS FOR RBT

FIGURE 15

References

[Carbonell 1983] Carbonell, J.G., "Learning by analogy: Formulation and generating plan from past experience," in *Machine Learning,* Michalski, Carbonell and Mitchell, Ed. Palo Alto, California: Tioga Press, 1983.

[Chow 1968] Chow, C.K. and C.N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transaction on Information Theory,* 1968, pp. 462-467.

[Dechter 1985] Dechter, R., UCLA, Los Angeles California, 1985. Phd thesis, in preparation.

[Even 1979] Even, S., *Graph Algorithms,* Maryland, USA: Computer Science Press, 1979.

[Freuder 1982] Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM,* Vol. 29, No. 1, 1982, pp. 24-32.

[Gaschnig 1979] Gaschnig, J., "A problem similarity approach to devising heuristics: first results," in *Proceedings 6th international joint conf. on Artificial Intelligence.,* Tokyo, Jappan: 1979, pp. 301-307.

[Guida 1979] Guida, G. and M. Somalvico, "A method for computing heuristics in problem solving," *Information Sciences,* Vol. 19, 1979, pp. 251-259.

[Haralick 1980] Haralick, R. M. and G.L. Elliot, "Increasing tree search efficiency for cconstraint satisfaction problems," *AI Journal,* Vol. 14, 1980, pp. 263-313.

[Knuth 1975] Knuth, D. E., "Esimating the efficiency of backtrack programs," *Mathematics of computation,* Vol. 29, No. 129, 1975, pp. 121-136.

[Mackworth 1977] Mackworth, A.K., "Consistency in networks of relations," *Artifficial intelligence,* Vol. 8, No. 1, 1977, pp. 99-118.

[Mackworth 1984] Mackworth, A.K. and E.C. Freuder, "The complexity of some polynomial consistancy algorithms for constraint satisfaction problems," *To appear in AI Journal* , 1984.

[Montanari 1974] Montanari, U., "Networks of constraints :fundamental properties and applications to picture processing," *information science,* Vol. 7, 1974, pp. 95-132.

[Nudel 1983] Nudel, B., "Consistent-Labeling problems and their algorithms: Expected complexities and theory based heuristics.," *Artificial Intelligence,* Vol. 21, 1983, pp. 135-178.

[Pearl 1983] Pearl, J., "On the discovery and generation of certain heuristics," *AI Magazine,* No. 22-23, 1983.

[Purdom 1985] Purdom, P.W. and C.A. Brown, *The Analysis of Algorithms*: CBS College Publishing, Holt, Rinehart and Winston, 1985.

[Sacerdoti 1974] Sacerdoti, E. D., "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence,* Vol. 5, No. 2, 1974, pp. 115-135.

[Simon 1975] Simon, H. A. and J. B. Kadane, "Optimal problem solving search: all or none solutions.," *Artificaial Intelligence,* Vol. 6, 1975, pp. 235-247.